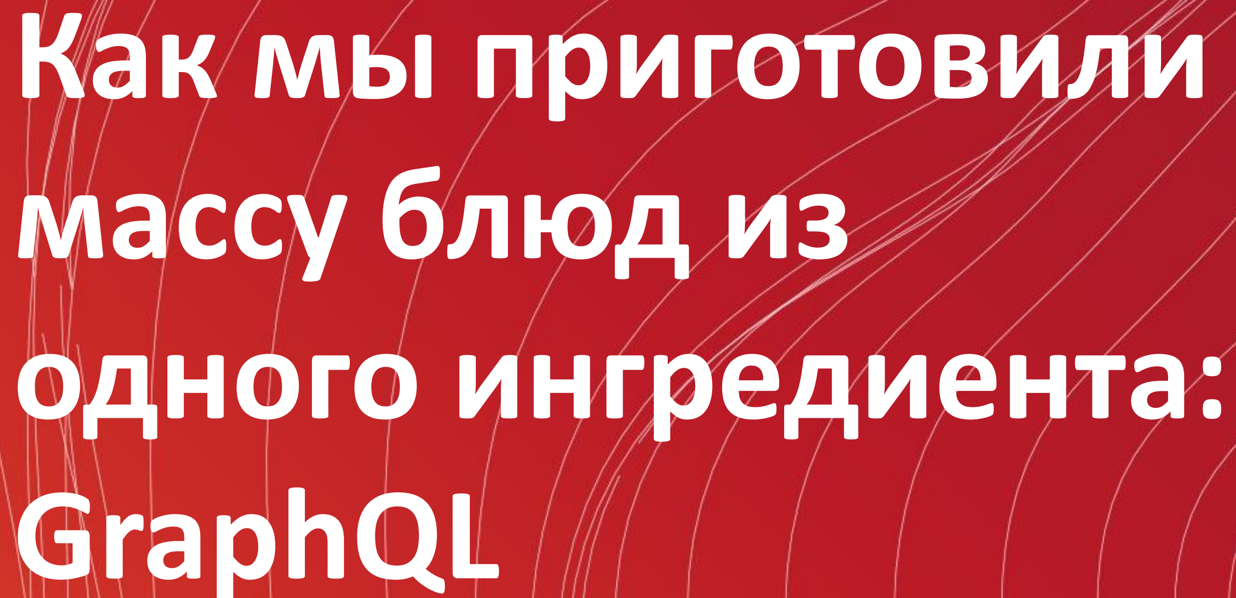


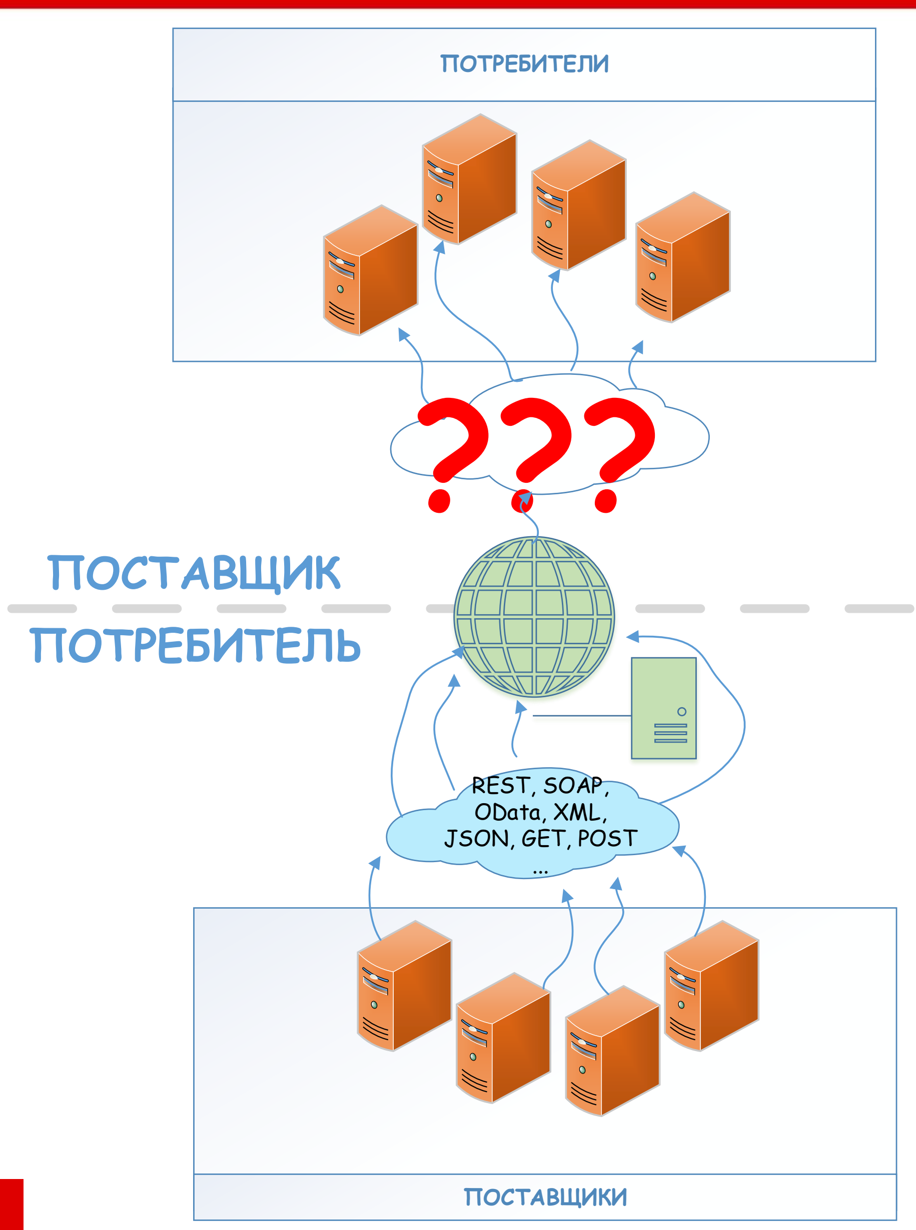
Как мы приготовили массу блюд из одного ингредиента: GraphQL

Сергей Тарасов, к.т.н.,
руководитель команд разработки,
Группа НЛМК





О чем пойдет речь в докладе



Цель:

Рассказать историю построения эффективного интеграционного взаимодействия с информационными системами на базе GraphQL



О чем пойдет речь в докладе

1. **Немного о нас**

2. **Потребитель VS поставщик**
    1. Потребитель
    2. Поставщик
    3. «Правила игры» поставщика
    4. Step by step VS One step
    5. Выводы

3. **GraphQL: внедрение и советы**
    1. Базовая структура
    2. Производительность: составной запрос (One step)
    3. Безопасность: Разграничение прав доступа (White list)
    4. Гибкость: Реализация сложных фильтров с логикой OR/AND

4. **Выводы**



НЛМК

1. Группа НЛМК

НЛМК США
Сталь
Прокат

НЛМК Калуга
Сталь
Прокат

НЛМК Дания
Прокат

НЛМК Европа (СП)
Сталь
Прокат

НЛМК Индия
Сервисный центр

СТОЙЛЕНСКИЙ ГОК
Концентрат
Окатыши

НЛМК Липецк
Кокс
Сталь
Прокат

ВИЗ-Сталь
Прокат

НЛМК Урал
Сталь
Прокат

Алтай-Кокс
Кокс

США
Дания
Бельгия
Италия
Франция
Россия
Москва
Липецк



>50 000 сотрудников
в России и за рубежом



>18 млн тонн
мощности по производству стали



1 место в России
по выпуску стали

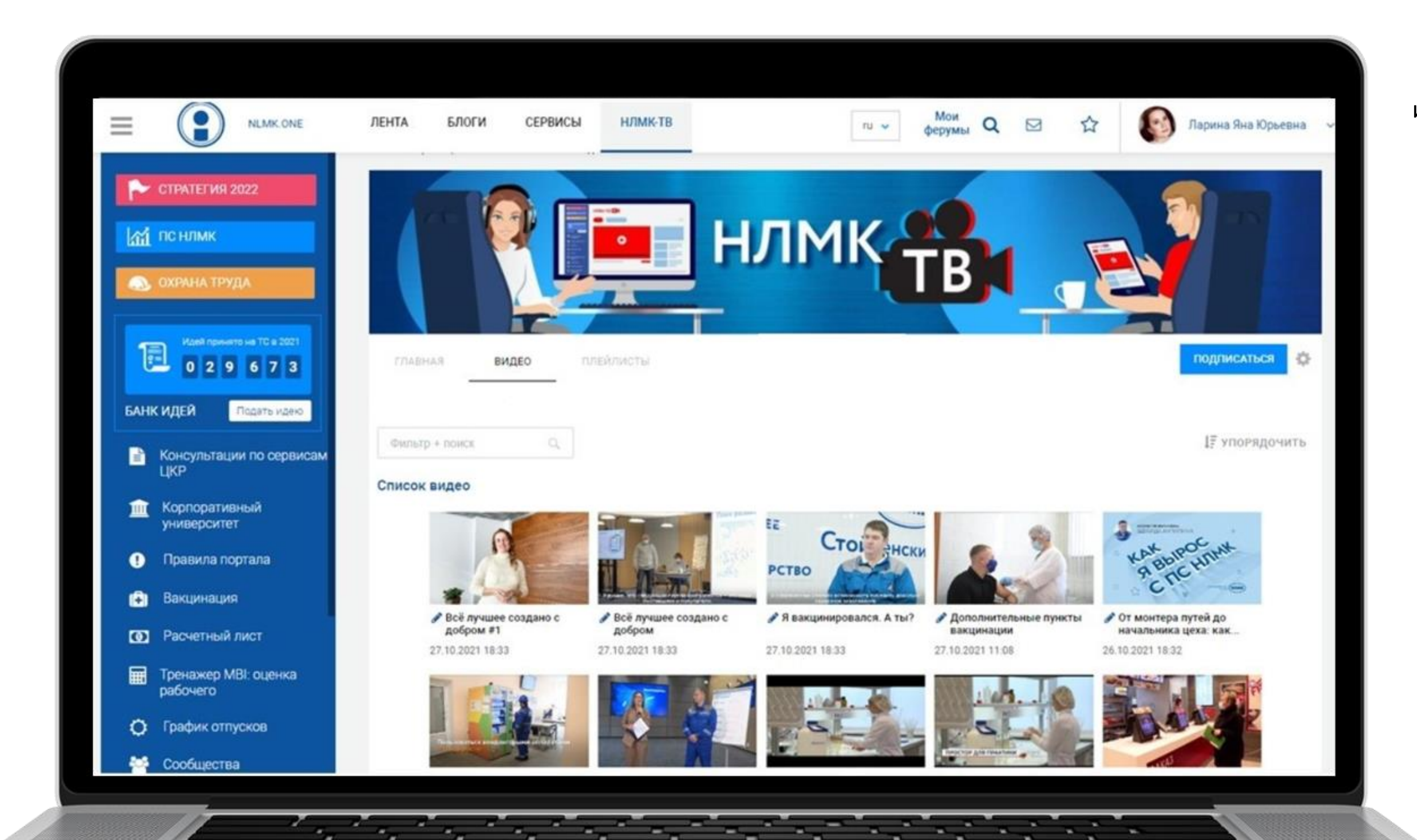


20 площадок в 7 странах
производственные активы





1. Единый корпоративный портал Группы НЛМК



>80 СЕРВИСОВ

из различных функциональных направлений со своей бизнес-логикой, механизмами интеграций, дорожной картой развития

>4 500 ДЕПАРТАМЕНТОВ

объединяет портал по орг. структуре

>50 000 СОТРУДНИКОВ,

объединенных в одной информационной системе

>100 ИНТЕРФЕЙСОВ

взаимодействия с другими системами

>15 СИСТЕМ,

с которыми интегрирован портал





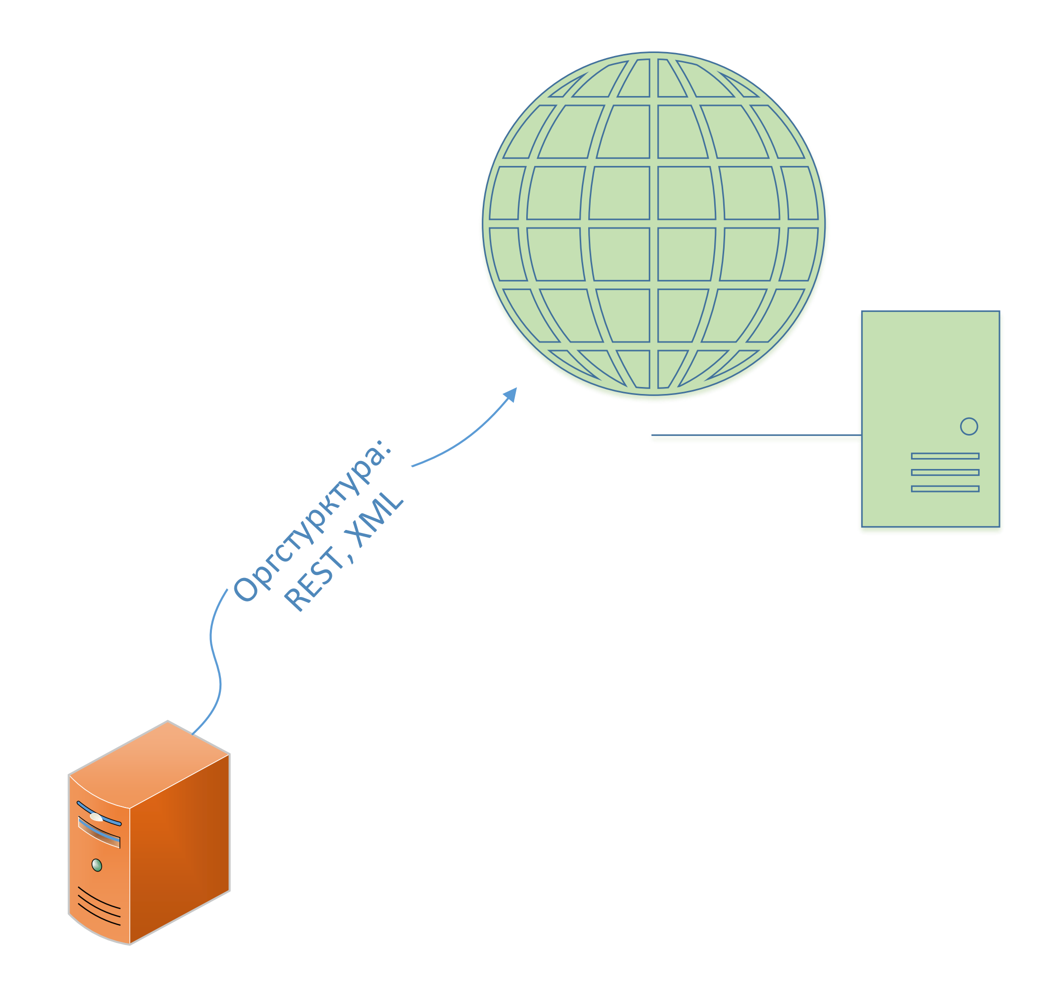
Раздел 2. Потребитель VS поставщик

1. **Потребитель**
2. **Поставщик**
3. **«Правила игры» поставщика**
4. **Step by step VS One step**





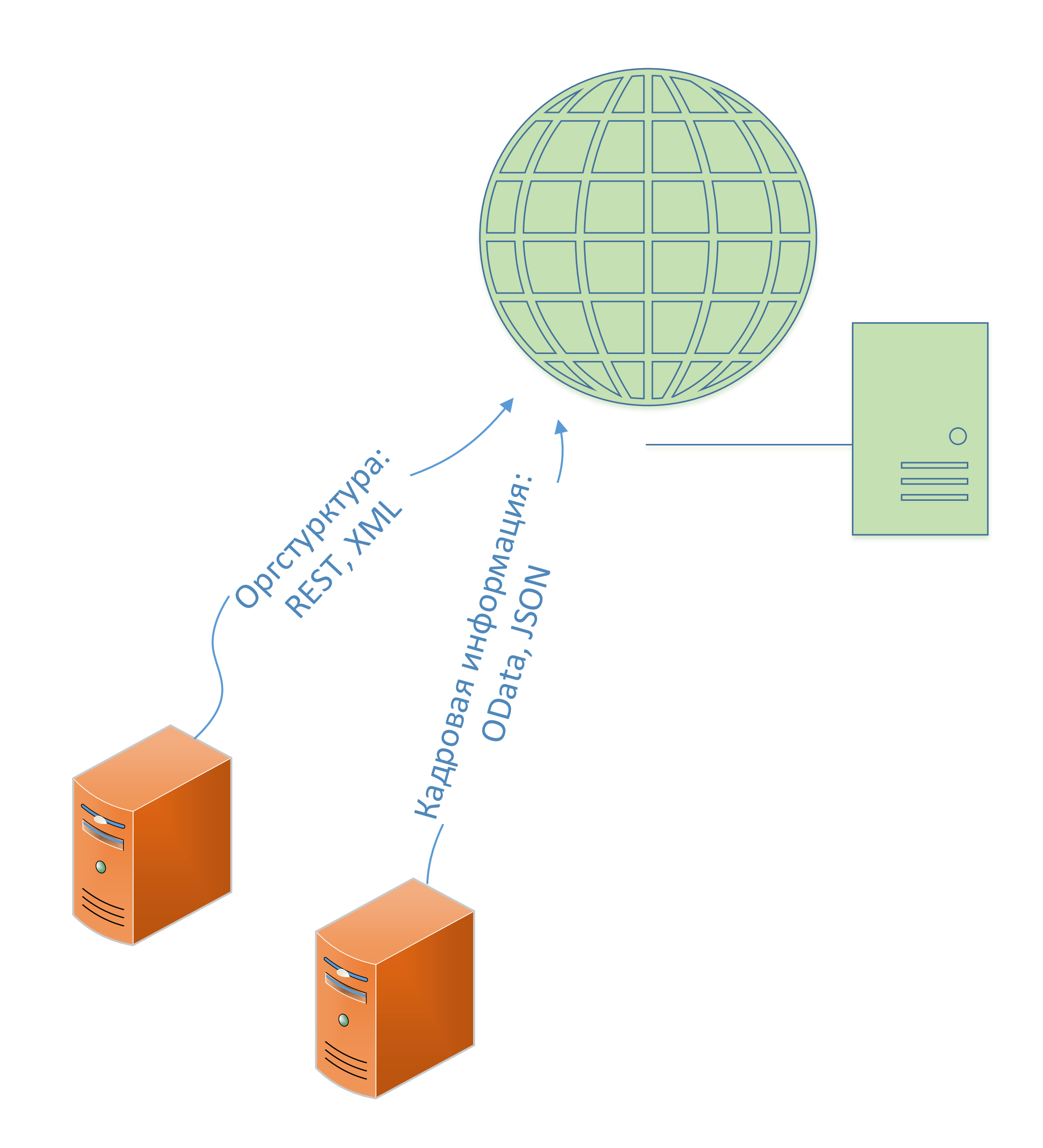
2.1. Потребитель





НЛМК

2.1. Потребитель





2.1. Потребитель



НЛМК

2.1. Потребитель

Оргстурктура: REST, XML

Кадровая информация: OData, JSON

Телефонный справочник: SOAP, XML

Расчетный лист: REST, JSON



2.1. Потребитель

ПОТРЕБИТЕЛЬ

REST, SOAP, OData, XML, JSON, GET, POST ...

ИС 1

ИС 2

ИС ..

ИС N

ПОСТАВЩИКИ



НЛМК

2.1. Потребитель

ПОТРЕБИТЕЛЬ

REST, SOAP,
OData, XML,
JSON, GET, POST
...

ИС 1
ИС 2
ИС ..
ИС N

ПОСТАВЩИКИ

ПОРТЕБИТЕЛЬ ЗНАЕТ:

- ✓ какая информация ему необходима и для чего
- ✓ поставщика с которым он будет работать
- ✓ язык/способ общения с поставщиком



НЛМК

2.1. Потребитель

ПОТРЕБИТЕЛИ

???

ПОСТАВЩИК

ПОТРЕБИТЕЛЬ

REST, SOAP,
OData, XML,
JSON, GET, POST
...

ПОСТАВЩИКИ



НЛМК

2.2. Поставщик

ПОТРЕБИТЕЛИ

?????

REST

SOAP

gRPC

OData

????

ПОСТАВЩИК



НЛМК

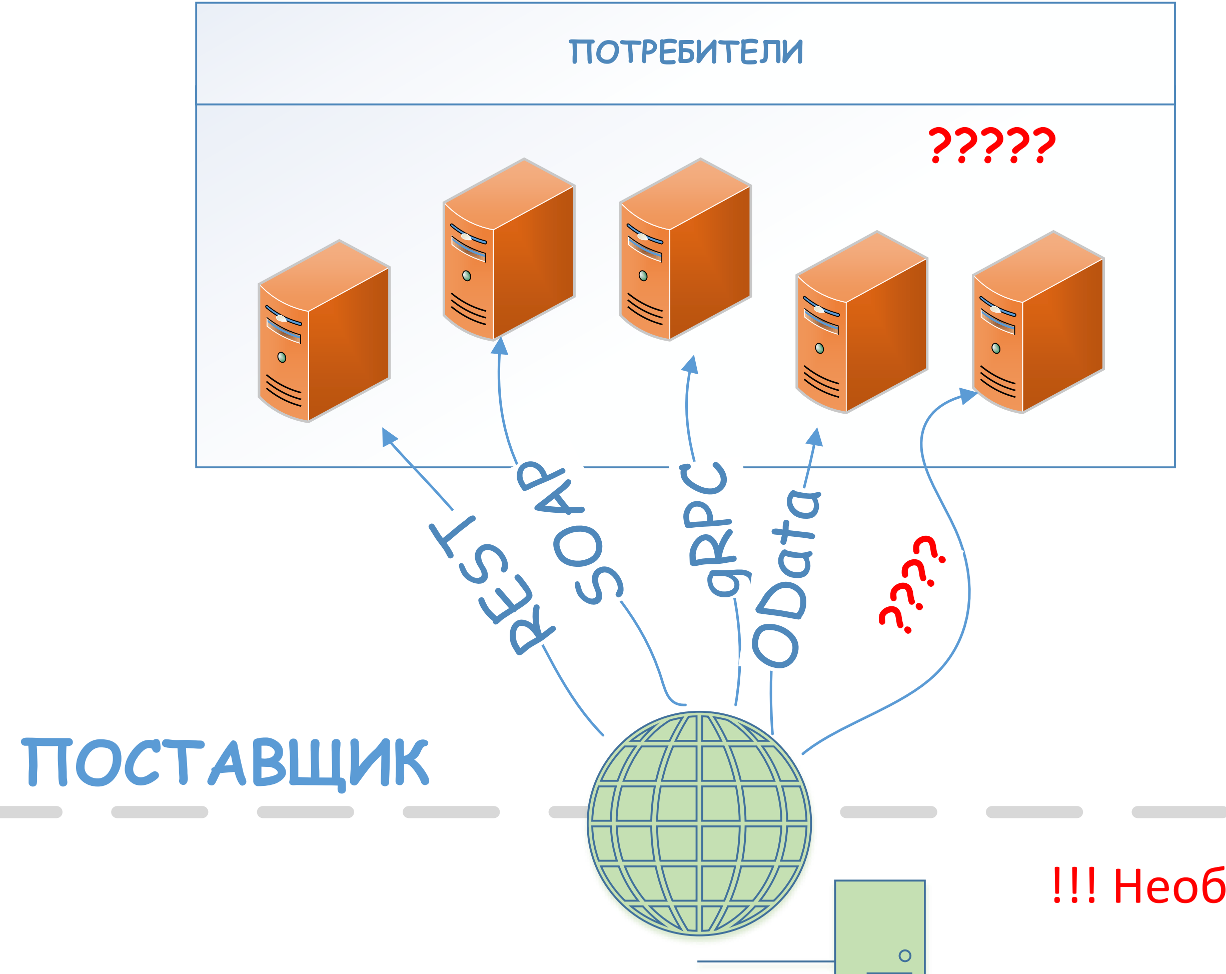
2.2. Поставщик

ПОСТАВЩИК НЕ ЗНАЕТ:

- ✓ количество возможных «потребителей»;
- ✓ платформу и технологический стек «потребителя»
- ✓ какая информация, ее состав (поля) и для чего она нужна «потребителю».



2.2. Поставщик



ПОСТАВЩИК НЕ ЗНАЕТ:

- ✓ количество возможных «потребителей»;
- ✓ платформу и технологический стек «потребителя»
- ✓ какая информация, ее состав (поля) и для чего она нужна «потребителю».

!!! Необходимы собственные «правила игры» !!!





2.3. «Правила игры» поставщика

Поставщик должен:

✓ **быть единообразным.** Работать одинаково для всех «потребителей» без исключения





2.3. «Правила игры» поставщика

Поставщик должен:

- ✓ **быть единообразным.** Работать одинаково для всех «потребителей» без исключения
- ✓ **работать по принципу One step**, т.е. за один шаг/запрос отдавать требуемую информацию (one-step)





2.4. Step by step VS One step

ВЕРХНЕУРОВНЕВАЯ ERP-ДИАГРАММА НОВОСТИ

Просмотры (views)
- SOURCE_ID
- USER_ID

Новость (news)
- PK ID
- NAME
- TEXT
- CREATE_DATE
- RUBRIC
- PUBLISH_DATE
- PREVIEW_TEXT
- AUTHOR

Пользователи (users)
- PK ID
- FIRST_NAME
- LAST_NAME

Рубрики (rubrics)
- PK ID
- NAME
- CODE

Комментарии (comments)
- PK ID
- TEXT
- AUTHOR
- DATE_CREATE
- SOURCE_ID

2.4. Step by step VS One step

Step-by-step



НЛМК

2.4. Step by step VS One step

One-step

Пользователь

Сервер приложений

БД

Вывод страницы новостей

Запрос news, comments, views, rubrics, users

Отдача news, comments, views, rubrics, users

Отдача страницы новостей





2.4. Step by step VS One step

min

Time

max

Step-by-step

Пользователь

Сервер приложений

БД

Вывод страницы новостей

Запрос news

Отдача news

Запрос comments

Отдача comments

Запрос views

Отдача views

Запрос rubrics

Отдача rubrics

Запрос users

Отдача users

Отдача страницы новостей

One-step

Пользователь

Сервер приложений

БД

Вывод страницы новостей

Запрос news, comments, views, rubrics, users

Отдача news, comments, views, rubrics, users

Отдача страницы новостей



2.4. Step by step VS One step

	Step by step	One step
Производительность	– **низкая** множество накладных расходов (сеть/транспорт, анализ запросов, формирование ответов и т.д.)	✓ **высокая**
Система управления знаниями	– **сложная** множества методов под каждый объект и их кросс-функциональное взаимодействие	✓ **простая** единый формат запроса/ответа
Порог вхождения	✓ **относительно невысокий** порог вхождения в архитектуру REST и написания «простых» http(s)-запросов	– **более высокий порог вхождения** требуется знать специфику работы умного endpoint'а





3. GraphQL: внедрение и советы



НЛМК

3. GraphQL: внедрение и советы

GraphQL – язык запросов с открытым исходным кодом.

Три основные характеристики:

- ✓ позволяет клиенту точно указать, какие данные ему нужны.
- ✓ облегчает агрегацию данных из нескольких источников.
- ✓ использует систему типов для описания данных.

GraphQL

users
subscribers
users
subscribers
likes
author
news
likes
rubrics
rubric
modify_by
users
user
count
text
users
author
comments
modify_by

В качестве основы мы используем библиотеку https://github.com/webonyx/graphql-php, которая соответствует изначально заложенной спецификации.





3. GraphQL: внедрение и советы

ПОТРЕБИТЕЛИ

GRAPHQL

ПОСТАВЩИК

ПОТРЕБИТЕЛЬ

REST, SOAP, OData, XML, JSON, GET, POST ...

ПОСТАВЩИКИ

GraphQL

users – subscribers – users
users – likes – news
users – subscribers – rubrics
news – author – users
users – likes – users
news – rubric – rubrics
news – modify_by – users
rubrics – user – users
news – count – users
news – text – comments
users – author – comments
comments – modify_by – users

PHP Russia 2022

НЛМК

3. GraphQL: внедрение и советы

1. **Базовая структура**

2. **Безопасность**
   Разграничение прав доступа (White list)

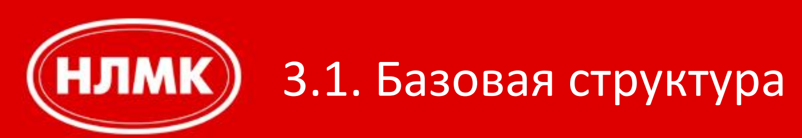
3. **Производительность**
   Составной запрос (One step)

4. **Гибкость**
   Реализация сложных фильтров с логикой OR/AND

GraphQL

users
subscribers
users
subscribers
likes
author
news
likes
rubrics
rubric
modify_by
users
user
count
text
modify_by
users
author
comments

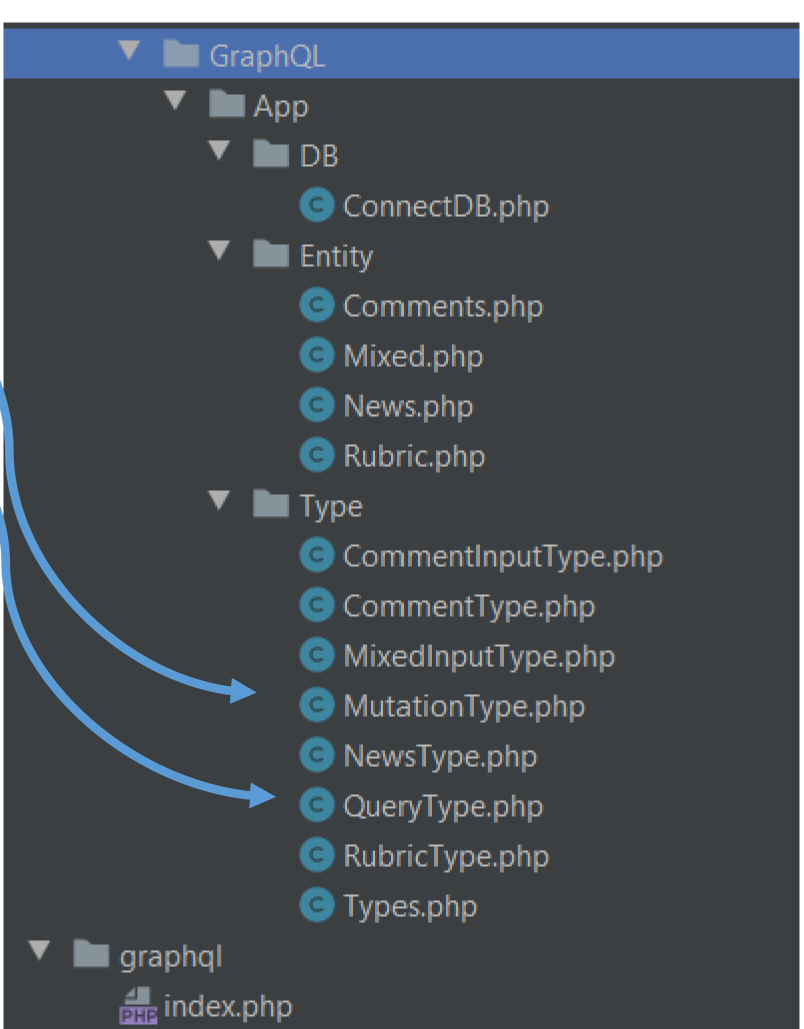




3.1. Базовая структура

В GraphQl API существует **2 корневых типа данных***:

- **MutationType** – используется для изменения информации
- **QueryType** – используется для получения информации



* Есть третий тип данных «Подписка», но он реализован только для «реактивных» библиотек PHP или PHP >8.1 (актуально для библиотеки https://github.com/webonyx/graphql-php)



3.1. Базовая структура

В GraphQl API существует **2 корневых типа данных***:

- **MutationType** – используется для изменения информации
- **QueryType** – используется для получения информации

Реестр «кастомных» типов данных для расширения простых типов данных.

```
▼ GraphQL
  ▼ App
    ▼ DB
        ConnectDB.php
    ▼ Entity
        Comments.php
        Mixed.php
        News.php
        Rubric.php
    ▼ Type
        CommentInputType.php
        CommentType.php
        MixedInputType.php
        MutationType.php
        NewsType.php
        QueryType.php
        RubricType.php
        Types.php
▼ graphql
    index.php
```

* Есть третий тип данных «Подписка», но он реализован только для «реактивных» библиотек PHP или PHP >8.1 (актуально для библиотеки https://github.com/webonyx/graphql-php)



НЛМК

3.1. Базовая структура

В GraphQl API существует **2 корневых типа данных***:

- **MutationType** – используется для изменения информации
- **QueryType** – используется для получения информации

Реестр «кастомных» типов данных для расширения простых типов данных.

Описание «кастомных» типов зарегистрированных в Types.php

```
GraphQL
  App
    DB
      ConnectDB.php
    Entity
      Comments.php
      Mixed.php
      News.php
      Rubric.php
    Type
      CommentInputType.php
      CommentType.php
      MixedInputType.php
      MutationType.php
      NewsType.php
      QueryType.php
      RubricType.php
      Types.php
  graphql
    index.php
```

* Есть третий тип данных «Подписка», но он реализован только для «реактивных» библиотек PHP или PHP >8.1 (актуально для библиотеки https://github.com/webonyx/graphql-php)



НЛМК

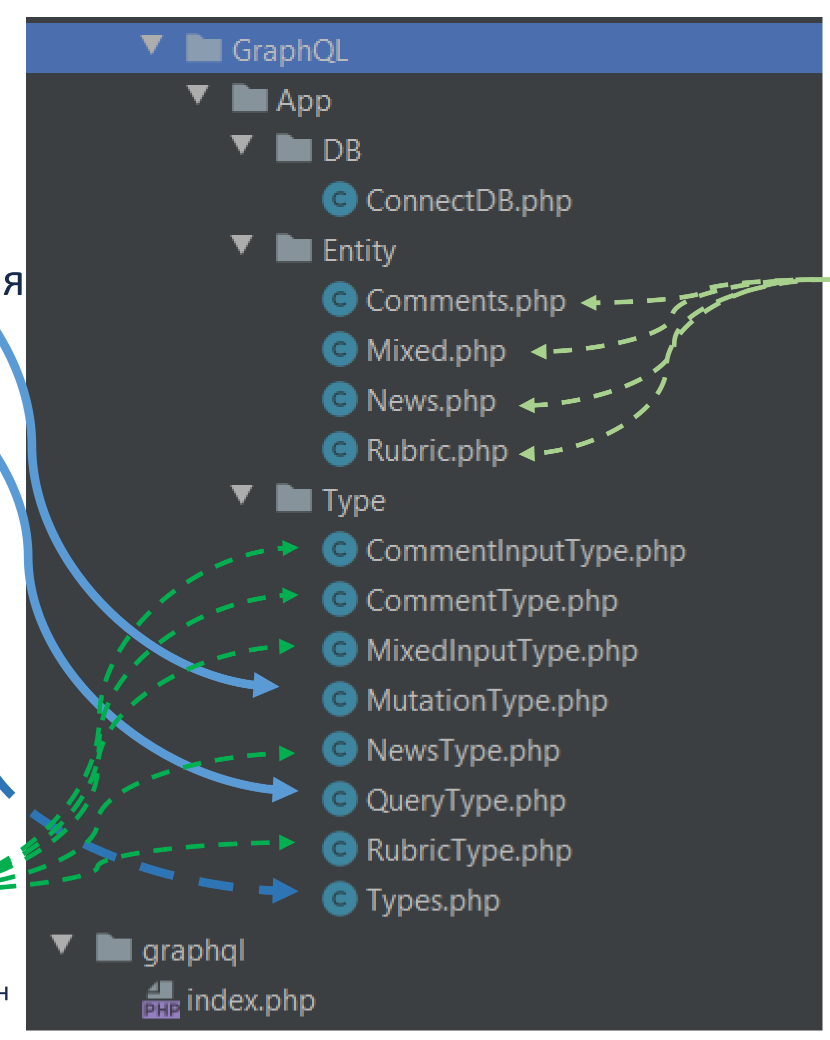
3.1. Базовая структура

В GraphQl API существует **2 корневых типа данных***:

- **MutationType** – используется для изменения информации
- **QueryType** – используется для получения информации

Реестр «кастомных» типов данных для расширения простых типов данных.

Описание «кастомных» типов зарегистрированных в Types.php



Сущности для реализации взаимодействия контроллера и модели данных

* Есть третий тип данных «Подписка», но он реализован только для «реактивных» библиотек PHP или PHP >8.1 (актуально для библиотеки https://github.com/webonyx/graphql-php)



НЛМК

3.1. Базовая структура

```
require_once realpath('../vendor/autoload.php');

try {
  $rawInput = file_get_contents('php://input');
  $input = json_decode($rawInput, true);
  $query = $input['query'];

  $schema = new Schema([
    'query' => Types::query(),
    'mutation' => Types::mutation()
  ]);

  $result = GraphQL::executeQuery($schema, $query);
  $arException = $result->toArray()['errors'] ?: [];

} catch (Throwable $e) {
  $result = ['error' => ['message' => $e->getMessage()]];
} finally {
  Log::push($query, $arException);
}
header('Content-Type: application/json; charset=UTF-8');
echo json_encode($result);
```

1

Определяем единую точку входа /graphql/index.php:

1. **Подключаем GraphQL**



НЛМК

3.1. Базовая структура

```
require_once realpath('../vendor/autoload.php');

try {
  $rawInput = file_get_contents('php://input');
  $input = json_decode($rawInput, true);
  $query = $input['query'];

  $schema = new Schema([
    'query' => Types::query(),
    'mutation' => Types::mutation()
  ]);

  $result = GraphQL::executeQuery($schema, $query);
  $arException = $result->toArray()['errors'] ?: [];

} catch (Throwable $e) {
  $result = ['error' => ['message' => $e->getMessage()]];
} finally {
  Log::push($query, $arException);
}
header('Content-Type: application/json; charset=UTF-8');
echo json_encode($result);
```

1

2

Определяем единую точку входа /graphql/index.php:

1. **Подключаем GraphQL**

2. **Получаем исходные данные**





3.1. Базовая структура


```php
require_once realpath('../vendor/autoload.php');

try {
  $rawInput = file_get_contents('php://input');
  $input = json_decode($rawInput, true);
  $query = $input['query'];

  $schema = new Schema([
    'query' => Types::query(),
    'mutation' => Types::mutation()
  ]);

  $result = GraphQL::executeQuery($schema, $query);
  $arException = $result->toArray()['errors'] ?: [];

} catch (Throwable $e) {
  $result = ['error' => ['message' => $e->getMessage()]];
} finally {
  Log::push($query, $arException);
}
header('Content-Type: application/json; charset=UTF-8');
echo json_encode($result);
```

1

2

3

Определяем единую точку входа /graphql/index.php:

1. **Подключаем GraphQL**
2. **Получаем исходные данные**
3. **Создаем схему**





3.1. Базовая структура

```
require_once realpath('../vendor/autoload.php');

try {
  $rawInput = file_get_contents('php://input');
  $input = json_decode($rawInput, true);
  $query = $input['query'];

  $schema = new Schema([
    'query' => Types::query(),
    'mutation' => Types::mutation()
  ]);

  $result = GraphQL::executeQuery($schema, $query);
  $arException = $result->toArray()['errors'] ?: [];

} catch (Throwable $e) {
  $result = ['error' => ['message' => $e->getMessage()]];
} finally {
  Log::push($query, $arException);
}
header('Content-Type: application/json; charset=UTF-8');
echo json_encode($result);
```

Определяем единую точку входа /graphql/index.php:

1. **Подключаем GraphQL**
2. **Получаем исходные данные**
3. **Создаем схему**
4. **Обрабатываем запрос**



НЛМК

3.1. Базовая структура

```
require_once realpath('../vendor/autoload.php');

try {
  $rawInput = file_get_contents('php://input');
  $input = json_decode($rawInput, true);
  $query = $input['query'];

  $schema = new Schema([
    'query' => Types::query(),
    'mutation' => Types::mutation()
  ]);

  $result = GraphQL::executeQuery($schema, $query);
  $arException = $result->toArray()['errors'] ?: [];

} catch (Throwable $e) {
  $result = ['error' => ['message' => $e->getMessage()]];
} finally {
  Log::push($query, $arException);
}
header('Content-Type: application/json; charset=UTF-8');
echo json_encode($result);
```

1

2

3

4

5

Определяем единую точку входа /graphql/index.php:

1. **Подключаем GraphQL**
2. **Получаем исходные данные**
3. **Создаем схему**
4. **Обрабатываем запрос**
5. **Обрабатываем ошибки**

PHP Russia 2022

НЛМК

3.1. Базовая структура

```
require_once realpath('../vendor/autoload.php');

try {
  $rawInput = file_get_contents('php://input');
  $input = json_decode($rawInput, true);
  $query = $input['query'];

  $schema = new Schema([
    'query' => Types::query(),
    'mutation' => Types::mutation()
  ]);

  $result = GraphQL::executeQuery($schema, $query);
  $arException = $result->toArray()['errors'] ?: [];

} catch (Throwable $e) {
  $result = ['error' => ['message' => $e->getMessage()]];
} finally {
  Log::push($query, $arException);
}
header('Content-Type: application/json; charset=UTF-8');
echo json_encode($result);
```

Определяем единую точку входа /graphql/index.php:

1. **Подключаем GraphQL**
2. **Получаем исходные данные**
3. **Создаем схему**
4. **Обрабатываем запрос**
5. **Обрабатываем ошибки**
6. **Производим логирование**



3.1. Базовая структура


```
require_once realpath('../vendor/autoload.php');

try {
  $rawInput = file_get_contents('php://input');
  $input = json_decode($rawInput, true);
  $query = $input['query'];

  $schema = new Schema([
    'query' => Types::query(),
    'mutation' => Types::mutation()
  ]);

  $result = GraphQL::executeQuery($schema, $query);
  $arException = $result->toArray()['errors'] ?: [];

} catch (Throwable $e) {
  $result = ['error' => ['message' => $e->getMessage()]];
} finally {
  Log::push($query, $arException);
}
header('Content-Type: application/json; charset=UTF-8');
echo json_encode($result);
```

Определяем единую точку входа /graphql/index.php:

1. **Подключаем GraphQL**
2. **Получаем исходные данные**
3. **Создаем схему**
4. **Обрабатываем запрос**
5. **Обрабатываем ошибки**
6. **Производим логирование**
7. **Выводим ответ**

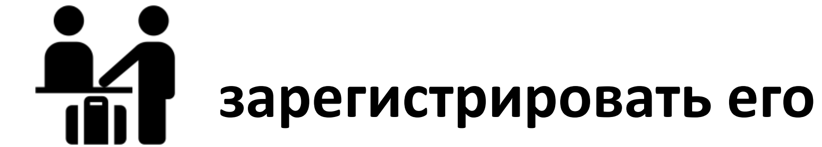
PHP Russia 2022

НЛМК

3.1. Базовая структура

Для создания типа, как правило, достаточно:













3.1. Базовая структура

Описываем запрос news

Конструктор класса "QueryType" содержит в себе все подтипы "query" запросов и он должен быть наследован от "ObjectType", что позволяет создавать свои составные типы данных.

```
class QueryType extends ObjectType
{
  public function __construct()
  {
    $config = [
      'fields' => function () {
        return [
          'news' => [ //Добавляем query запрос news
            'type' => Types::news(), // Создаем новый тип данных
            'args' => [
              'id' => Types::int(), // Добавляем аргумент, по которому мы можем получить новость
            ],
            'resolve' => function ($root, $args, $context, $info) {
              return News::get($args); // Добавляем метод get для получения результата запроса
            }
          ]
        ];
      }
    ];
    parent::__construct($config);
  }
}
```

НЛМК

3.1. Базовая структура

Регистрация типа news

```
class Types extends Type
{
  private static $query;
  private static $news;
  private static $rubric;

  public static function query()
  {
    return self::$query ?: (self::$query = new QueryType());
  }

  public static function news() // Зарегистрировали новый тип
  {
    return self::$news ?: (self::$news = new NewsType()); //необходимо описать поля этого типа
  }
}
```






3.1. Базовая структура


```
class NewsType extends ObjectType
{
  public function __construct()
  {
    $config = [
      'fields' => function () {
        return [ //описали поля таблицы и указали их типы
          'id' => [
            'type' => Types::int(),
          ],
          'name' => [
            'type' => Types::string(),
          ],
          'text' => [
            'type' => Types::string(),
          ],
        ];
      }
    ];
    parent::__construct($config);
  }
}
```



Описание возвращаемых полей для типа news



3.1. Базовая структура

```
namespace NLMK\GraphQL\App\Type;

use NLMK\GraphQL\App\DB\ConnectDB;

class News
{
  public static function get($args)
  {
    $DB = new ConnectDB();
    return $DB->query("SELECT * from news WHERE id = {$args['id']}")[0];
  }
}
```



Описание получения данных из БД для новости





3.1. Базовая структура

GraphQL: проверка работы

headers [2]

```
1 {"query":"{news (id: 6130) {id, name, text}}"}
```

Text JSON XML HTML | Format body | Enable body evaluation length: 46 bytes

Response Cache Detected - Elapsed Time: 151ms

200 OK

BODY pretty

headers [14]


```
{
  data: {
    news: [
      {
        id: 6130,
        name: "Бронза из Магнитогорска",
        text: "<p></p><div align=\"justify\"><div align=\"justify\"><p><
      }
    ]
  }
}
```


Top Bottom Collapse Open 2Request Copy Download





3.1. Базовая структура

GraphQL: проверка работы

BODY

Text

```
1 {"query":"{news (id: 6130) {id, name}}"}
```

Text JSON XML HTML | Format body | Enable body evaluation

length: 40 bytes

Cache Detected - Elapsed Time: 122ms

BODY

pretty

```
{
  data: {
    news: [
      {
        id: 6130,
        name: "Бронза из Магнитогорска"
      }
    ]
  }
}
```

Top Bottom Collapse Open 2Request Copy Download





3.2. Производительность: составной запрос (One step)

Описываем запрос rubric

```
class QueryType extends ObjectType
{
  public function __construct()
  {
    $config = [
      'fields' => function () {
        return [
          'news' => [/*...*/],
          'rubric' => [ //Добавили еще один запрос
            'type' => Types::rubric(), // Создаем новый тип данных - рубрика
            'args' => [
              'id' => Types::int(), // Добавили аргумент, по которому мы можем получить рубрику
            ],
            'resolve' => function ($root, $args, $context, $info) {
              return Rubric::get($args); // Добавляем метод get для получения результата запроса
            }
          ]
        ];
      }
    ];
    parent::__construct($config);
  }
}
```



3.2. Производительность: составной запрос (One step)

Регистрация типа rubric

```
class Types extends Type
{
  private static $query;
  private static $news;
  private static $rubric;

  public static function query()
  {
    return self::$query ?: (self::$query = new QueryType());
  }

  public static function news()
  {
    return self::$news ?: (self::$news = new NewsType());
  }

  public static function rubric() // Зарегистрировали новый тип
  {
    return self::$rubric ?: (self::$rubric = new RubricType()); // необходимо описать поля этого типа
  }
}
```






3.2. Производительность: составной запрос (One step)



Описание возвращаемых полей для типа rubric

```
class RubricType extends ObjectType
{
  public function __construct()
  {
    $config = [
      'fields' => function () {
        return [ //описали поля таблицы и указали их типы
          'id' => [
            'type' => Types::int(),
          ],
          'name' => [
            'type' => Types::string(),
          ],
          'code' => [
            'type' => Types::string(),
          ],
        ];
      }
    ];
    parent::__construct($config);
  }
}
```






3.2. Производительность: составной запрос (One step)



Описание получения данных из БД для рубрик

```
namespace NLMK\GraphQL\App\Type;

use NLMK\GraphQL\App\DB\ConnectDB;

class Rubric
{
  public static function get($args)
  {
    $DB = new ConnectDB();
    return $DB->query("SELECT * from rubrics WHERE id = {$args['id']}")[0];
  }
}
```






3.2. Производительность: составной запрос (One step)

Описание возвращаемых полей для типа news

```
class NewsType extends ObjectType
{
  public function __construct()
  {
    $config = [
      'fields' => function () {
        return [
          'id' => [
            'type' => Types::int(),
          ],
          'name' => [
            'type' => Types::string(),
          ],
          'text' => [
            'type' => Types::string(),
          ],
          'rubric' => [
            'type' => Types::Rubric(), //добавляем тип для возврата id рубрик
          ],
        ];
      }
    ];
    parent::__construct($config);
  }
}
```






3.2. Производительность: составной запрос (One step)

headers [2]

```
{"query":"{news (id: 6130) {id, name, text, rubric}}"}
```

Text JSON XML HTML | Format body | Enable body evaluation length: 54 bytes

Response

Cache Detected - Elapsed Time: 159ms

200 OK

BODY pretty

headers [14]


```
{
  data: {
    news: [
      {
        id: 6130,
        name: "Бронза из Магнитогорска",
        text: "<p></p><div align=\"justify\"><div align=\"justify\"><p><
        rubric: 18
      }
    ]
  }
```


Top Bottom Collapse Open 2Request Copy Download





3.2. Производительность: составной запрос (One step)

headers [2]

```
1 {"query":"{news (id: 6130) {id, name, text, rubric}}"}
```

Text JSON XML HTML | Format body | Enable body evaluation length: 54 bytes

Response

Cache Detected - Elapsed Time: 159ms

200 OK

BODY pretty

headers [14]

```
{
  data: {
    news: [
      {
        id: 6130,
        name: "Бронза из Магнитогорска",
        text: "<p></p><div align=\"justify\"><div align=\"justify\"><p><
        rubric: 18
      }
    ]
  }
```

Top Bottom Collapse Open 2Request Copy Download

Нам вернулся корректный ответ, но

чтобы получить рубрику новости, необходимо послать 2 запроса (Step-by-step):

1. news ~ 159 ms





3.2. Производительность: составной запрос (One step)

BODY

Text

```
{"query":"{rubric (id: 18) {id, name, code}}"}
```

Text JSON XML HTML | Format body | Enable body evaluation length: 46 bytes

headers [2]

Response

Cache Detected - Elapsed Time: 121ms

200 OK

BODY

pretty

headers [14]

```
{
  data: {
    rubric: [
      {
        id: 18,
        name: "НЛМК PRO",
        code: "nlmkpro"
      }
    ]
  }
```

Top Bottom Collapse Open 2Request Copy Download

Нам вернулся корректный ответ, но

чтобы получить рубрику новости, необходимо послать 2 запроса (Step-by-step):

1. news ~ 159 ms
2. rubric ~121 ms

Итого: ~280 ms





3.2. Производительность: составной запрос (One step)

BODY

```
{"query":"{rubric (id: 18) {id, name, code}}"}
```

Text JSON XML HTML | Format body | Enable body evaluation — length: 46 bytes

Response — Cache Detected - Elapsed Time: 121ms

200 OK

BODY (pretty)

```
{
  data: {
    rubric: [
      {
        id: 18,
        name: "НЛМК PRO",
        code: "nlmkpro"
      }
    ]
  }
}
```

Top Bottom Collapse Open 2Request Copy Download

Нам вернулся корректный ответ, но

чтобы получить рубрику новости, необходимо послать 2 запроса (Step-by-step):

1. news ~ 159 ms
2. rubric ~121 ms

Итого: ~280 ms

Реализуем составной запрос, чтобы мы могли получить эти же данные при одном запросе (One-step).





3.2. Производительность: составной запрос (One step)

Реализуем составной запрос

```
class NewsType extends ObjectType
{
  public function __construct()
  {
    $config = [
      'fields' => function () {
        return [
          'id' => ['type' => Types::int()],
          'name' => ['type' => Types::string()],
          'text' => ['type' => Types::string()],
          'rubric' => [  //описали вложенный запрос на получение рубрики новости
            'type' => Types::rubric(),
            //передаем в метод параметры для отбора рубрик по родительским возвращаемым полям
            'resolve' => function ($root, $args, $context, $info) {
              return Rubric::get(['id' => $root['rubric']]);
            }
          ],
        ];
      }
    ];
    parent::__construct($config);
  }
}
```



3.2. Производительность: составной запрос (One step)

headers [2]

```
1 {"query":"{news (id: 6130) {id, name, text, rubric{id, name, code}}}"}
```

Text JSON XML HTML | Format body | Enable body evaluation length: 70 bytes

Response

Cache Detected - Elapsed Time: 161ms

200 OK

headers [14]

BODY pretty

```
{
  data: {
    news: [
      {
        id: 6130,
        name: "Бронза из Магнитогорска",
        text: "<p></p><div align=\"justify\"><div align=\"justify\"><p><
        rubric: {
          id: 18,
          name: "НЛМК PRO",
          code: "nlmkpro"
        }
      }
    ]
  }
```

Top Bottom Collapse Open 2Request Copy Download

Нам вернулся корректный ответ по новости и рубрике за один шаг (One step)

Итого: ~161 ms





3.2. Производительность: составной запрос (One step)

headers [2]


```
1 {"query":"{news (id: 6130) {id, name, text, rubric{id, name, code}}}"}
```


Text JSON XML HTML | Format body | Enable body evaluation   length: 70 bytes

Response   Cache Detected - Elapsed Time: 161ms

200 OK

headers [14]   BODY   pretty


```
{
  data: {
    news: [
      {
        id: 6130,
        name: "Бронза из Магнитогорска",
        text: "<p></p><div align=\"justify\"><div align=\"justify\"><p><
        rubric: {
          id: 18,
          name: "НЛМК PRO",
          code: "nlmkpro"
        }
      }
    ]
  }
```


Top Bottom Collapse Open 2Request Copy Download

Нам вернулся корректный ответ по новости и рубрике за один шаг (One step)

Итого: ~161 ms

	Step by step	One step
Производительность	~280 ms	~161 ms – 57,5%





3.3. Безопасность: Разграничение прав доступа (White list)





3.3. Безопасность: Разграничение прав доступа (White list)

Добавление проверки в корневой тип данных

```
class QueryType extends ObjectType
{
  public function __construct()
  {
    $arConfig = [ //Вынесли наши Query запросы в массив
      'news' => [
        'type' => Types::listOf(Types::news()),
        'args' => [
          'id' => Types::int(),
        ],
        'resolve' => function ($root, $args, $context, $info) {
          return News::get($args);
        }
      ],
      'rubric' => [],
    ];
    $config = [
      'fields' => function () use ($arConfig) {
        //проверка доступов к методам и его полям
        return Permission::processConfigFields($arConfig);
      }
    ];
    parent::__construct($config);
  }
```






3.3. Безопасность: Разграничение прав доступа (White list)

Структура конфигурационных данных

```
namespace GraphQL\App\Auth; //файл App\Auth\Permission.php

use GraphQL\Server\RequestError;

class Permission
{
  private static $envConfig = [
    [
      "LOGIN" => "TEST",
      "PASSWORD" => "123456",
      "WHITELIST" => [
        "news" => ["id", "name", "text"] //разрешенный метод и возвращаемые поля для пользователя
      ]
    ],
    [
      "LOGIN" => "RUBRIC_USER",
      "PASSWORD" => "654321",
      "WHITELIST" => [
        "rubric" => ["id"]
      ]
    ]
}
```

*Рекомендуется выносить данные параметры за пределы корневой директории сайта (DOCUMENT_ROOT)



3.3. Безопасность: Разграничение прав доступа (White list)

Метод проверки данных

```
class Permission
{
  //....................................
  public static function processConfigFields($fields)
  {
    foreach ($fields as $code => $arField) {

      $fields[$code]['resolve'] = function ($root, $args, $context, $info) use ($code, $arField) {

        $result = $arField['resolve']($root, $args, $context, $info);
        //проверка доступа
        self::checkAccess($code, array_keys(current($result)));
        //в случае успеха возвращаем результат
        return $result;

      };
    }

    return $fields;
  }
  //....................................
}
```



3.3. Безопасность: Разграничение прав доступа (White list)

Метод проверки доступа к переданному типу запроса и его аргументам.

```
class Permission
{
  /** Метод проверки доступа к переданному типу запроса и его аргументам */
  private static function checkAccess(string $methodName, array $args)
  {
    if (!empty($_SERVER['PHP_AUTH_USER'])) {
      foreach (self::$envConfig as $arUser) {
        //Проверим, разрешен ли запрос для пользователя
        if ($_SERVER['PHP_AUTH_USER'] == $arUser['LOGIN'] &&
            $_SERVER['PHP_AUTH_PW'] == $arUser['PASSWORD']) {
          if (!in_array($methodName, array_keys($arUser['WHITELIST']))) {
            throw new RequestError($methodName . ' is forbidden for user ' . $arUser['LOGIN'], 401);
          }
          //Проверим, разрешены ли возвращаемые поля запроса для пользователя
          foreach ($args as $field) {
            if ($arUser['WHITELIST'][$methodName] &&
                !in_array($field, $arUser['WHITELIST'][$methodName])) {
              throw new RequestError($field . ' is forbidden for user ' . $arUser['LOGIN'], 401);
            }
          }
  ...........
  }
}
```



3.3. Безопасность: Разграничение прав доступа (White list)

Метод проверки доступа к переданному типу запроса и его аргументам.

```
class Permission
{
  /** Метод проверки доступа к переданному типу запроса и его аргументам */
  private static function checkAccess(string $methodName, array $args)
  {
    if (!empty($_SERVER['PHP_AUTH_USER'])) {
      foreach (self::$envConfig as $arUser) {
        //Проверим, разрешен ли запрос для пользователя
        if ($_SERVER['PHP_AUTH_USER'] == $arUser['LOGIN'] &&
            $_SERVER['PHP_AUTH_PW'] == $arUser['PASSWORD']) {
          if (!in_array($methodName, array_keys($arUser['WHITELIST']))) {
            throw new RequestError($methodName . ' is forbidden for user ' . $arUser['LOGIN'], 401);
          }
          //Проверим, разрешены ли возвращаемые поля запроса для пользователя
          foreach ($args as $field) {
            if ($arUser['WHITELIST'][$methodName] &&
                !in_array($field, $arUser['WHITELIST'][$methodName])) {
              throw new RequestError($field . ' is forbidden for user ' . $arUser['LOGIN'], 401);
            }
          }
  ...........
  }
}
```






3.3. Безопасность: Разграничение прав доступа (White list)

headers [2]

```
{"query":"{news (id: 6130) {id, name, text, rubric{id, name, code}}}"}
```

Text JSON XML HTML | Format body | Enable body evaluation length: 70 bytes

Response

Cache Detected - Elapsed Time: 106ms

200 OK

BODY pretty

headers [14]

```
{
  errors: [
    {
      message: "Method news is forbidden for user TEST",
      category: "graphql",
      locations: [
        {
          line: 1,
          column: 2
        }
      ],
      path: [
        "news"
      ]
    }
  ],
```

Top Bottom Collapse Open 2Request Copy Download



3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND





3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Необходимо:



создать новый тип данных (MixedInputType). Который будет прототипом для автоматической генерации классов со сложной логикой и добавлением в Query-запросов налету.



зарегистрировать новый тип данных



добавить логику обработки в тип Query запросов



3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Класс должен автоматически генерировать/расширять все типы данных для реализации сложных фильтров с типами “OR”/”AND” в режиме реалтайма, **НО**



3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Класс должен автоматически генерировать/расширять все типы данных для реализации сложных фильтров с типами “OR”/”AND” в режиме реалтайма, **НО**

Библиотека не работает с одноименными классами данных и выдает ошибку.

«Schema must contain unique named types but contains multiple types named "\GraphQL\App\Type\Input\MixedInputType\" (see http://webonyx.github.io/graphql-php/type-system/#type-registry).»



3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Класс должен автоматически генерировать/расширять все типы данных для реализации сложных фильтров с типами “OR”/”AND” в режиме реалтайма, **НО**

Библиотека не работает с одноименными классами данных и выдает ошибку.

«Schema must contain unique named types but contains multiple types named "\GraphQL\App\Type\Input\MixedInputType\" (see http://webonyx.github.io/graphql-php/type-system/#type-registry).»

Для обхода ограничения необходимо добавлять «соль» при автогенерации новых типов данных





3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Новый тип данных MixedInputType

```
class MixedInputType extends InputObjectType
{
  public function __construct(array $arExtendedFields = [])
  {
    $sHashGenerated = md5(implode(',', $arExtendedFields));//создаем хеш аргументов

     $config = [
      //автоформирование схемы типов, генерация уникального имени для нового типа, например,
      //GraphQL\App\Type\Input\MixedInputType05afd6ecb065cfd7b660a6b6a59a54cf
      'name' => self::class . $sHashGenerated,

      //опишем автоформирование полей для query запросов
      'fields' => function () use ($arExtendedFields, $sHashGenerated) {

      }
    ];
    parent::__construct($config);
  }
}
```






3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Новый тип данных MixedInputType

```
class MixedInputType extends InputObjectType
{
  public function __construct(array $arExtendedFields = [])
  {
    $sHashGenerated = md5(implode(',', $arExtendedFields));//создаем хеш аргументов

     $config = [
      //автоформирование схемы типов, генерация уникального имени для нового типа, например,
      //GraphQL\App\Type\Input\MixedInputType05afd6ecb065cfd7b660a6b6a59a54cf
      'name' => self::class . $sHashGenerated,

      //опишем автоформирование полей для query запросов
      'fields' => function () use ($arExtendedFields, $sHashGenerated) {

      }
    ];
    parent::__construct($config);
  }
}
```






3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Новый тип данных MixedInputType

```
class MixedInputType extends InputObjectType
{
  public function __construct(array $arExtendedFields = [])
  {
    $sHashGenerated = md5(implode(',', $arExtendedFields));//создаем хеш аргументов

     $config = [
      //автоформирование схемы типов, генерация уникального имени для нового типа, например,
      //GraphQL\App\Type\Input\MixedInputType05afd6ecb065cfd7b660a6b6a59a54cf
      'name' => self::class . $sHashGenerated,

      //опишем автоформирование полей для query запросов
      'fields' => function () use ($arExtendedFields, $sHashGenerated) {

      }
    ];
    parent::__construct($config);
  }
}
```






3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND




```
'fields' => function () use ($arExtendedFields, $sHashGenerated) {
  return [//автоформирование_полей_для_Query_запросов
    'OR' => [
      'type' => Types::listOf(
        new InputObjectType(
          [
            'name' => self::class . 'OR' . $sHashGenerated,
            'fields' => function () use ($arExtendedFields) {return $arExtendedFields;}
          ]
        )
      )
    ],
  ];
}
```






3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND




```
'fields' => function () use ($arExtendedFields, $sHashGenerated) {
  return [//автоформирование полей для Query запросов
    'OR' => [
      'type' => Types::listOf(
        new InputObjectType(
          [
            'name' => self::class . 'OR' . $sHashGenerated,
            'fields' => function () use ($arExtendedFields) {return $arExtendedFields;}
          ]
        )
      )
    ],
    'AND' => [/*аналогично ‘OR’ за исключением поля name – вместо ‘OR’ указать ‘AND’*/]
  ];
}
```






3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Новый тип данных MixedInputType

```
class MixedInputType extends InputObjectType
{
  public function __construct(array $arExtendedFields = [])
  {
    $sHashGenerated = md5(implode(',', $arExtendedFields));//создаем хеш аргументов
    $config = [
      'name' => self::class . $sHashGenerated,//автоформирование схемы типов, генерация
уникального имени для нового типа, например,
GraphQL\App\Type\Input\MixedInputType05afd6ecb065cfd7b660a6b6a59a54cf
      'fields' => function () use ($arExtendedFields, $sHashGenerated) {
        return [//автоформирование полей для Query запросов
          'OR' => [
            'type' => Types::listOf(
              new InputObjectType(
                [
                  'name' => self::class . 'OR' . $sHashGenerated,
                  'fields' => function () use ($arExtendedFields) {return $arExtendedFields;}
                ]
              )
            )
          ],
          'AND' => [/*аналогично ‘OR’ за исключением поля name – вместо ‘OR’ указать ‘AND’*/]
        ];
      }
    ];
    parent::__construct($config);

}
```






3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Регистрация нового типа данных

```
class Types extends Type
{
  private static $query;
  private static $mixedInput;

  public static function query()
  {
    return self::$query ?: (self::$query = new QueryType());
  }

  /**Смешанный тип данных для query запроса*/
  public static function mixedInput(array $arExtendedFields = [])
  {
    $sArguments = md5(implode(',', $arExtendedFields));
    return self::$mixedInput[$sArguments] ?: self::$mixedInput[$sArguments] = new
MixedInputType($arExtendedFields);
  }
}
```






3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Внедрение логики обработки

```
class QueryType extends ObjectType
{
  public function __construct()
  {
    $arConfig = [//Вынесли наши Query запросы в массив
      'news' => [
        'type' => Types::listOf(Types::news()),
        'args' => ['id' => Types::int()],
        'resolve' => function ($root, $args, $context, $info) {return News::get($args);}
      ],
    ];
    $config = [//формируем фильтры со сложной логикой для каждого типа данных
      'fields' => function () use ($arConfig) {
        foreach ($arConfig as $sCodeConfig => $arParams) {
          $arConfig[$sCodeConfig]['args']['filter'] =
Types::listOf(Types::mixedInput($arConfig[$sCodeConfig]['args']));
        }
        return $arConfig;
      }
    ];
    parent::__construct($config);
  }
}
```

PHP Russia 2022



3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

Ручная генерация

```
class QueryType extends ObjectType
{
    public function __construct()
    {
        $arConfig = [
            'news' => [
                'type' => Types::listOf(Types::news()),
                'args' => [
                    'id' => Types::int(),
                    'code' => Types::string(),
                    'filter' => [ //повторяющийся аргумент "filter" с логикой OR\AND для типа news
                        'name' => 'logicFilterNews',
                        'fields' => function () {
                            return [
                                'OR' => [
                                    'type' => Types::listOf(
                                        new InputObjectType(
                                            [
                                                'name' => self::class . 'OR' . 'news',
                                                'fields' => function () {
                                                    return [ //повторяющиеся аргументы фильтра для типа news
                                                        'id' => Types::int(),
                                                        'code' => Types::string(),
                                                    ];
                                                }
                                            ]
                                        )
                                    )
                                ],
                                'AND' => [
                                    //................................................
                                ]
                            ];
                        }
                    ],
                ],
                'resolve' => function ($root, $args, $context, $info) {
                    return News::get($args);
                }
            ],
        ];

        $config = [
            'fields' => function () use ($arConfig) {
                return $arConfig;
            }
        ];

        parent::__construct($config);
```

Автоматическая генерация

```
class QueryType extends ObjectType
{
    public function __construct()
    {
        $arConfig = [   //Вынесли наши Query запросы в массив
            'news' => [
                'type' => Types::listOf(Types::news()),
                'args' => [
                    'id' => Types::int(),
                ],
                'resolve' => function ($root, $args, $context, $info) {
                    return News::get($args);
                }
            ],
        ];

        $config = [
            'fields' => function () use ($arConfig) {
                //формируем фильтры со сложной логикой для каждого типа данных
                foreach ($arConfig as $sCodeConfig => $arParams) {
                    $arConfig[$sCodeConfig]['args']['filter'] =
Types::listOf(Types::mixedInput($arConfig[$sCodeConfig]['args']));
                }
                return $arConfig;
            }
        ];

        parent::__construct($config);
    }
}
```



3.4. Гибкость: Реализация сложных фильтров с логикой OR/AND

BODY

Text

```
1 {"query":"{news (filter: {OR: [{id: 6130},{id: 6137}]}) {id, name, text, rubric{id, name, code}}}"}
```

headers [2]

Text JSON XML HTML | Format body | Enable body evaluation

length: 99 bytes

Response

Cache Detected - Elapsed Time: 194ms

200 OK

BODY

pretty

headers [14]

```
{
  data: {
    news: [
      {id: 6137, name: "С чего начиналась история? Выпуск 13", text: "<p>В <a href=
      {id: 6130, name: "Бронза из Магнитогорска", text: "<p></p><div align=\"justif
    ]
  }
}
```

lines nums copy

length: 42 kilobytes

Top Bottom Collapse Open 2Request Copy Download





4. Выводы

- ✓ Мы описали >20 «простых» типов данных
- ✓ В среднем 1 «комплексный» тип данных состоит из 3-х подтипов

Итого: нам удалось реализовать >1 000 комбинаций/сочетаний.

$$C_n^k = \frac{n!}{(n-k)! * k!},$$

где n – множество/количество объектов,

k – количество сочетаний объекта

$$C_{20}^3 = \frac{20!}{(20-3)! * 3!} = 1140$$



✓**Определите назначение системы** в разрезе поставщик/потребитель

✓**Реализовывайте принцип One step-запросов**

✓**Заложите систему разграничения доступа** к объектам и их сущностям для каждого потребителя

✓**Настройте систему логирования** с четкой фиксацией потребителя, который запрашивает информацию

✓**Сделать фильтр** со сложной логикой для большей гибкости запросов

GraphQL

users
subscribers
users
likes
author
subscribers
news
likes
rubrics
rubric
modify_by
users
user
count
text
modify_by
users
author
comments





Оцените доклад



Сергей Тарасов, к.т.н.,
руководитель команд разработки,
Группа НЛМК

Email: tsergei2012@gmail.com
TG: @tsergei2012

Презентация: bit.ly/3VlbkqBq

